6-1-2015

GPU-accelerated lip-tracking library

Alex DeBoui
Santa Clara University

Jesse Harder
Santa Clara University

SANTA CLARA UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

Date: June 1, 2015

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Alex DeBoni
Jesse Harder

ENTITLED

GPU-Accelerated Lip-Tracking Library

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

Thesis Advisor

Department Chair

GPU-Accelerated Lip-Tracking Library

by

Alex DeBoni
Jesse Harder

Submitted in partial fulfillment of the requirements
for the degree of
Bachelor of Science in Computer Science and Engineering
School of Engineering
Santa Clara University

Santa Clara, California
June 1, 2015

GPU-Accelerated Lip-Tracking Library

Alex DeBoni
Jesse Harder

Department of Computer Engineering
Santa Clara University
June 1, 2015

ABSTRACT

A major part of having correct pronunciation when learning a new language is moving your lips in the correct way. This is a difficult thing to learn and to teach. One solution to this is software which tracks a student's lip movements and provides feedback. This paper describes how we have created a C++ library to accurately track lips in provided images. Further, this library attempts to use a CUDA-enabled GPU implementation to improve the algorithm's performance. It will fall back on a CPU implementation if such a GPU is not found. As a result, the lip tracking library runs on Windows, Linux, and OS X, as well as Android devices.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Background

A major part of learning a new language is to be able to pronounce words correctly. To be able to pronounce words correctly, a student's mouth needs to make proper movements. This is difficult to teach in a classroom setting, especially when a student needs to repeat the same phrase many times. The student can become frustrated with the teacher and not want to continue, especially if he or she feels as though he or she is being viewed as incompetent by his or her peers. However, when using a computer, the student can feel more comfortable trying the same phrase over and over without embarrassment.

Currently, there are desktop and mobile applications to help with a student's pronunciation by listening to him or her speak and providing feedback. If the student does not make the correct mouth shape, however, he or she is far less likely to pronounce the word correctly. Applications that only analyze audio can, at best, only guess what the student might be doing incorrectly with his or her mouth so these applications may not provide the help the student really needs.

1.2 Solution

Our solution to this problem is a mobile application to help a student pronounce words correctly by not only listening to him or her speak and analyzing the audio, but also by watching his or her mouth and providing feedback for both the audio and video information. We have created an algorithm to accurately track lip movement. A generic tracking algorithm will not work for lip motion tracking since mouths change shape too dramatically when they move. For this reason the lip tracking algorithm that we have implemented is be specialized to mouth contours. The data provided from this software is then combined with audio input to analyze how the student may improve his or her pronunciation. For example, the application can tell the student if he or she held

his or her mouth open too long or didn't open his or her mouth enough for a particular word. In the end, this tool would make it easier and faster for students learning a new language to master correct pronunciation.

Chapter 2

Requirements

We had several requirements that needed to be met in order to implement our desired system. These requirements can be divided into two categories. Functional requirements are those which our system must do. Generally, they can be evaluated only as either true or false, depending on whether or not the system actually does or does not do what it should. Nonfunctional requirements are those which describe the manner in which the functional requirements should be achieved. They are evaluated based on a degree of satisfaction. Additionally, our system also has a couple design constraints, which are limits on the design and implementation of the system. The following requirements were to be met for successful implementation of our lip-tracking library.

2.1 Functional

The system will:

- Receive camera images from the user.
- Return a point array of lip edges.

2.2 Nonfunctional

The system will be:

- Efficient - it will run quickly enough for use in a real time system.
- User Friendly - it will have an easy to use interface.
- Reliable/accurate - it will produce correct data.
- Robust - it won't crash. (i.e. Its crash frequency will be within acceptable norms.)
- Reusable - it will be easy to use in other systems.

2.3 Design Constraints

The system must:

- Be able to run on Android devices and desktop computers.
- Use CUDA.

Chapter 3

Conceptual Model

The interface to our lip-tracking library consists of six functions. The first is the `initializeTracker` function, which takes as a parameter the file containing the shape data that the tracker uses to perform its analysis. This will allow the tracking process to start. The next two functions, `getAcceptanceThreshold` and `setAcceptanceThreshold`, allow the user to get and set a threshold value used in creating a lip contour. This threshold value determines the required confidence in the lip-tracking algorithm. The lower the threshold value, the less accurate the algorithm is. If the value is too high, however, the algorithm may fail to obtain a lip contour at the desired confidence level, resulting in an error being reported. The `getLipContour` function is used to obtain a lip contour model as an array of points. This function takes as parameters the file path to the image file to be processed and a reference to the array in which the lip contour information is to be stored. It also returns a status code indicating success or failure. The array that is passed in must have already been allocated by the user. To do this, the user will use the `getNumberOfContourPoints` function, which returns the size that the array should be. Finally, the user may also call the `resetTracker` function to tell the tracking process to re-evaluate the entire image instead of just the region in which it believes the lips to be.

3.1 API Functions

- `void initializeTracker(char *iniFile);`
- `int getAcceptanceThreshold();`
- `void setAcceptanceThreshold(int threshold);`
- `int getNumberOfContourPoints();`
- `int getLipContour(char *filePath, float contour[]);`
- `void resetTracker();`

Chapter 4

Use Cases

Based on our conceptual model, our system has a number of actions that users might perform. Any action that a user might take can be considered a use case, which is defined as the steps needed to achieve a goal on the application. Presented below are the use cases for our library, which effectively correspond to the different functions in the library. Under each use case, the actor, goal, precondition, postcondition, sequence of steps, and exceptions are listed. All the use cases of our application are shown below in Figure 4.1. Following the diagram are detailed descriptions of each use case.

4.1 Use Case 1: Initialize Tracker

Actor: User.

Goal: Initialize the tracking program with the shape data used for tracking.

Preconditions: User has imported our library (which includes shape data).

Postconditions: Tracker is ready to accept images.

Sequence of Steps: User calls the `initializeTracker` with shape data filepath.

4.2 Use Case 2: Get Acceptance Threshold

Actor: User.

Goal: Obtain a value for the acceptance threshold for the lip tracking process.

Preconditions: User has imported our library.

Postconditions: User has acceptance threshold value.

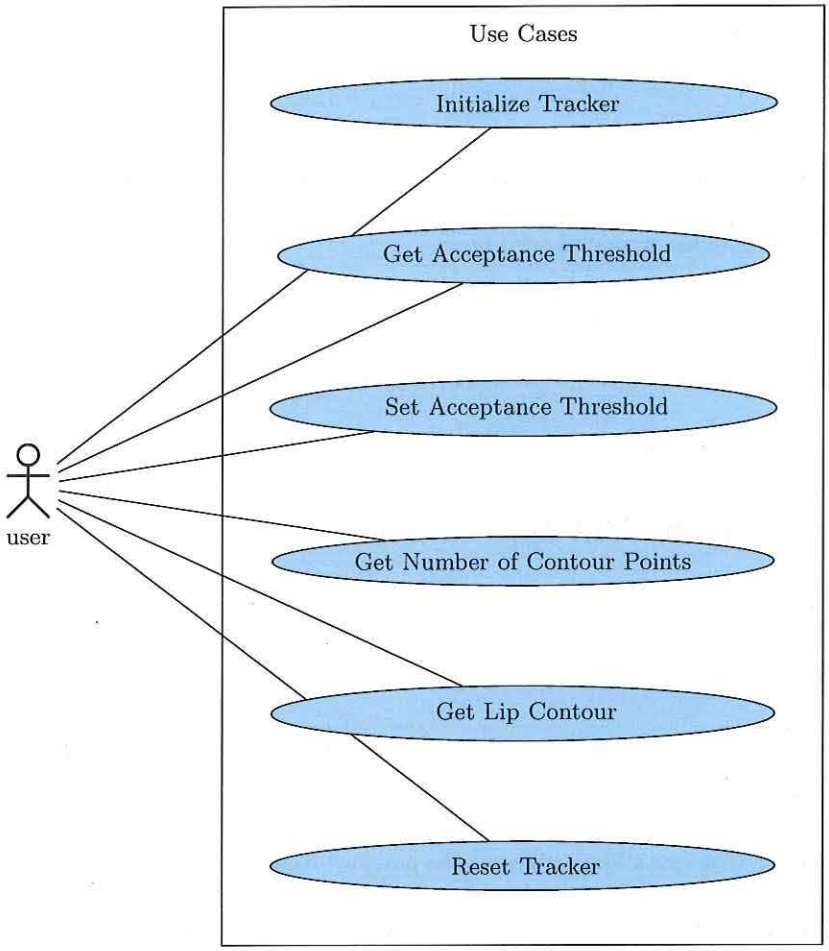

Figure 4.1: Use case diagram

Sequence of Steps: User calls the `getAcceptanceThreshold` function.

Exceptions: None.

4.3 Use Case 3: Set Acceptance Threshold

Actor: User.

Goal: Change the value for the acceptance threshold for the lip tracking process.

Preconditions: User has imported our library.

Postconditions: Acceptance threshold value is changed to provided value.

Sequence of Steps: User calls the `setAcceptanceThreshold` function.

4.4 Use Case 4: Get Number of Contour Points

Actor: User.

Goal: Obtain the number of contour points provided by the lip-tracking process.

Preconditions: User has imported our library.

Postconditions: User has the number of contour points.

Sequence of Steps: User calls the `getNumberOfContourPoints` function.

Exceptions: None.

4.5 Use Case 5: Get Lip Contour

Actor: User.

Goal: Obtain lip contour data from a face image.

Preconditions: User has imported our library, has an image to process, and has allocated an array of the proper size.

Postconditions: User has a lip contour for the provided image.

Sequence of Steps: User calls the `getLipContour` function on filepath to the facial image.

Exceptions: The image provided is not of a face: the function will return an error code.

4.6 Use Case 6: Reset Tracker

Actor: User.

Goal: Re-establish accurate tracking of lips.

Preconditions: User has imported our library.

Postconditions: Tracker is accurately tracking lips in provided images.

Sequence of Steps: User calls the `resetTracker` function.

Chapter 5

Technologies Used

Below we list all of the technologies that we used in our system and the basic reasons why we chose to use each of them.

5.1 C++

We used C++ because it is a relatively efficient programming language. Additionally, it allowed us to create one library that can be cross-compiled and reused on any platform. Further, both of us have a significant amount of experience with the language.

5.2 CUDA

CUDA (Compute Unified Device Architecture) is a parallel computing platform and programming model developed by Nvidia for their GPUs. We used CUDA because Android devices are now starting to get dedicated GPU chips that are optimized for CUDA.

5.3 Nvidia Tegra K1

We used the Nvidia Tegra K1 GPU because it is the currently the most common mobile GPU being marketed, with phones like the Google Nexus 9 utilizing it.

5.4 OpenCV

We used the OpenCV library to do image processing such as noise filtering and edge detection. OpenCV also detects if there is a CUDA-capable GPU on the system and will use it if it finds one. Additionally, it is compatible with Windows, Linux, OS X, Android, and iOS, so it will work for all of our implementations.

5.5 CMake

CMake is a cross-platform make utility which detects what compilers are installed on the system and selects an appropriate to use for the project. This made it easy for us to test changes on different platforms by streamlining the compilation process.

Chapter 6

Design Rationale

In this chapter we list and explain the reasoning for the design choices we have made for the lip-tracking library.

6.1 Technologies

A description of the technologies we used can be found in Chapter 5. We chose these technologies because they are very common and well supported, in addition to being available on every platform with which we are working (Windows, Linux, OS X, Android).

6.2 Application Programming Interface

We wanted to make our API as simple as possible for the user, while still being flexible. As a result, we allow images to be inputted in many common and popular formats. The full list of compatible image formats is as follows:

Bitmaps `*.bmp, *.dib`

JPEG files `*.jpeg, *.jpg, *.jpe`

JPEG 2000 files `*.jp2`

Portable Network Graphics `*.png`

Portable image format `*.pbm, *.pgm, *.ppm`

Sun rasters `*.sr, *.ras`

TIFF files `*.tiff, *.tif`

The user can also change the acceptance threshold for lip detection. If there are too many false positives, then the threshold can be lowered to compensate, and vice versa.

6.3 Algorithm

There are many different algorithms that have been used in the past for feature tracking. These can be divided into three categories, geometric-based approaches, appearance-based approaches, and model-based approaches. Ahmad Hassanat has provided a literature review comparing these approaches in his paper *Visual Speech Recognition* [4].

6.3.1 Geometric-Based Approaches

These approaches involve edge detection and then trying to match ratios of the edge's height, width, area and perimeter with some predefined values.

6.3.2 Appearance-Based Approaches

These approaches involve color segmentation of the image and then trying to match a segment with a predefined set of colors. This is the most common approach currently taken [6][8], but does not always have good results. Increasing the contrast beforehand can somewhat improve the results.

6.3.3 Model-Based Approaches

These approaches involve creating a statistical model of a feature shape by inputting a set of hand-landmarked images into a neural network. This model can then be used to very accurately find a similar shape in an image. It does this by creating a small image patch around each landmark point in the model and then iteratively fits the patches to the target image. There are three common model-based approaches, the Active Contour Model (ACM), Active Shape Model (ASM), and Active Appearance Model (AAM). These three models are closely related in that they all use the same fundamental algorithm. However, ASM is an improvement over ACM and AAM is an improvement over ASM. These improvements bring more accuracy to the algorithm, however, they also increase complexity and runtime.

Additionally, both ASM [5] and AAM [7] have been shown to be parallelizable with CUDA.

6.3.4 Our Approach

We chose to attempt a hybrid approach that combines all three previous approaches. The geometric-based and appearance-based approaches alone do not produce results that are accurate enough for us. They run very quickly, however, and we believed that they could produce a better input image for a model-based approach. The results from our attempt at using the geometric-based and appearance-based approaches can be found in Chapter 11. The model-based approach we are using is the Active

Shape Model. We are using an Active Shape Model implementation that is detailed in the book *Mastering OpenCV* [1]. The model is a good balance between the Active Contour Model, which is too simple and not accurate enough, and the Active Appearance Model, which is complicated and relatively slow. We also convert all of the input images to grayscale as this speeds up many of the processing operations. An example of all the operations we planned to perform can be found in Figure 6.1. Finally, another optimization this algorithm has is that it uses a Haar Cascade (a data structure that generalizes the shape of the object we're looking for) on the first frame of the detection. The cascade narrows down the search area quickly and then in subsequent frames, we only search the area that the lips were last found in.

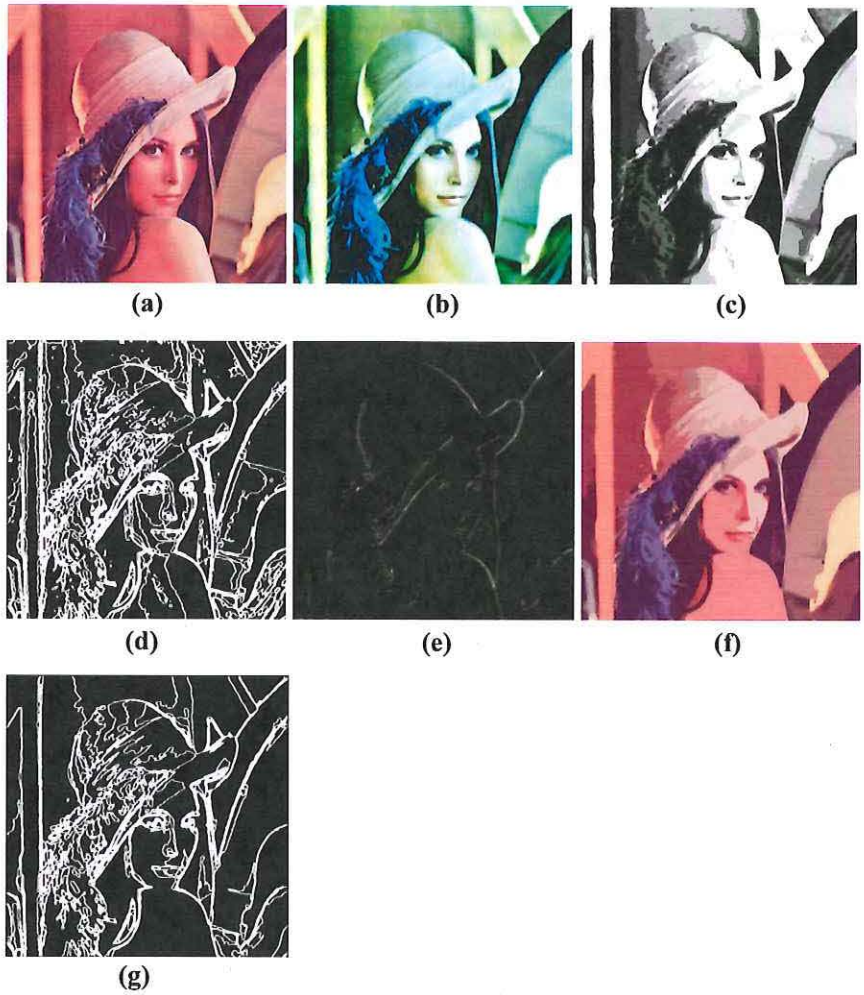

Figure 6.1: (a) Source image
(b) Increased contrast
(c) Grayscale
(d) Heavy edge detection
(e) Light edge detection
(f) Color segmentation
(g) Medium edge detection
Figure taken from Zhang[9]

Chapter 7

Architectural Diagram

We used a data-flow architecture for our system, also known as a pipe-and-filter architecture. The system has an array of pixel values for an image which gets modified by each filter it passes through.Our original plan for the data flow for our system can be seen in Figure 7.1. The current data flow of our system more closely resembles Figure 7.2.

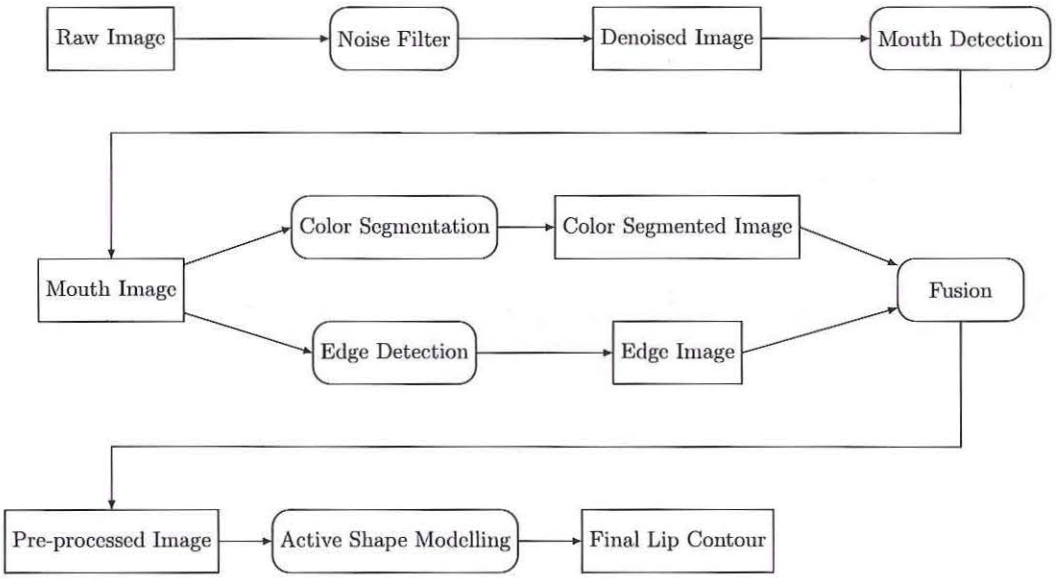

Figure 7.1: Planned system architecture

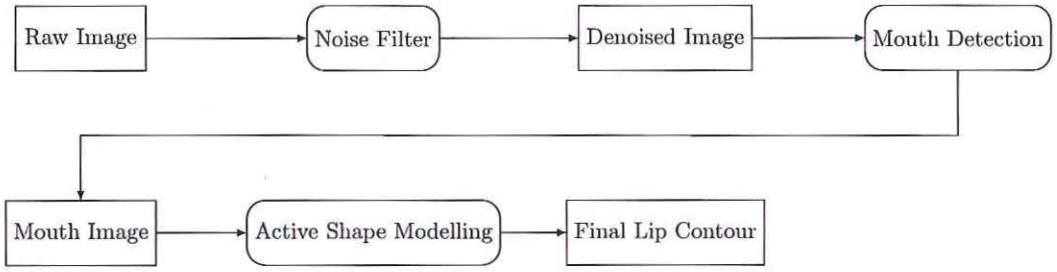

Figure 7.2: Current system architecture

Chapter 8

Test Plan

It was necessary for us to test our system in order to ensure that our lip-tracking library was fully functional and met all of the requirements we identified for it. There were a number of different ways that we tested our system, the main points of which are listed below.

8.1 Alpha Testing

We have a library of videos and images of people's lips. We visually analyzed the output of our program for these inputs to see how accurately it tracked the lips. We also did runtime testing to evaluate the speed of different parts of the code in order to improve overall efficiency.

8.2 Beta Testing

We got several peer volunteers with a diverse set of facial features to test our library with a webcam and visually analyzed the results. The variables we aimed to test include mustaches, beards, teeth presence, lip color, and skin color. Beyond our peer volunteers, we got a number of volunteer testers when we attended and demoed our product at this years GPU Technology Conference.

Chapter 9

Project Risks

Table 9.1 below shows the potential risks we identified for our project. Listed with each risk are the consequences should the risk occur, the probability of the risk occurring, the severity of the risk, the impact of the risk (calculated by multiplying probability and severity), and the mitigation strategies we employed to avoid the risk from occurring.

Risk	Consequences	Probability	Severity	Impact	Mitigation
Poor time management	Not having a finished product.	0.4	7	2.8	Follow gantt chart schedule. Prioritize features of system.
GPU transition problems	Delays. Potentially only CPU implementation finished.	0.3	8	2.4	Consider GPU implementation while working on CPU version.
Team member becomes sick or otherwise disabled	Loss of time.	0.4	4	1.6	Stay informed on each other's progress.Soft deadlines.
Major issues with chosen technologies	Loss of time and progress.	0.2	7	1.4	Early testing of technologies. Consider alternative technologies in advance.
Failure to gather sufficient beta testers	Incomplete testing.	0.5	2	1.0	Scout beta testers in advance. Have thorough alpha testing plan.

Table 9.1: Risk analysis table

Chapter 10

Development Timeline

The figures below show timelines of the work we have done as well as when we will be working on and finishing each major task of this project that we have yet to complete, including the writing of the library, finishing the documentation, and preparation for the final presentations in the spring. By following this plan, we have and will continue to keep ourselves on schedule for our upcoming deadline.

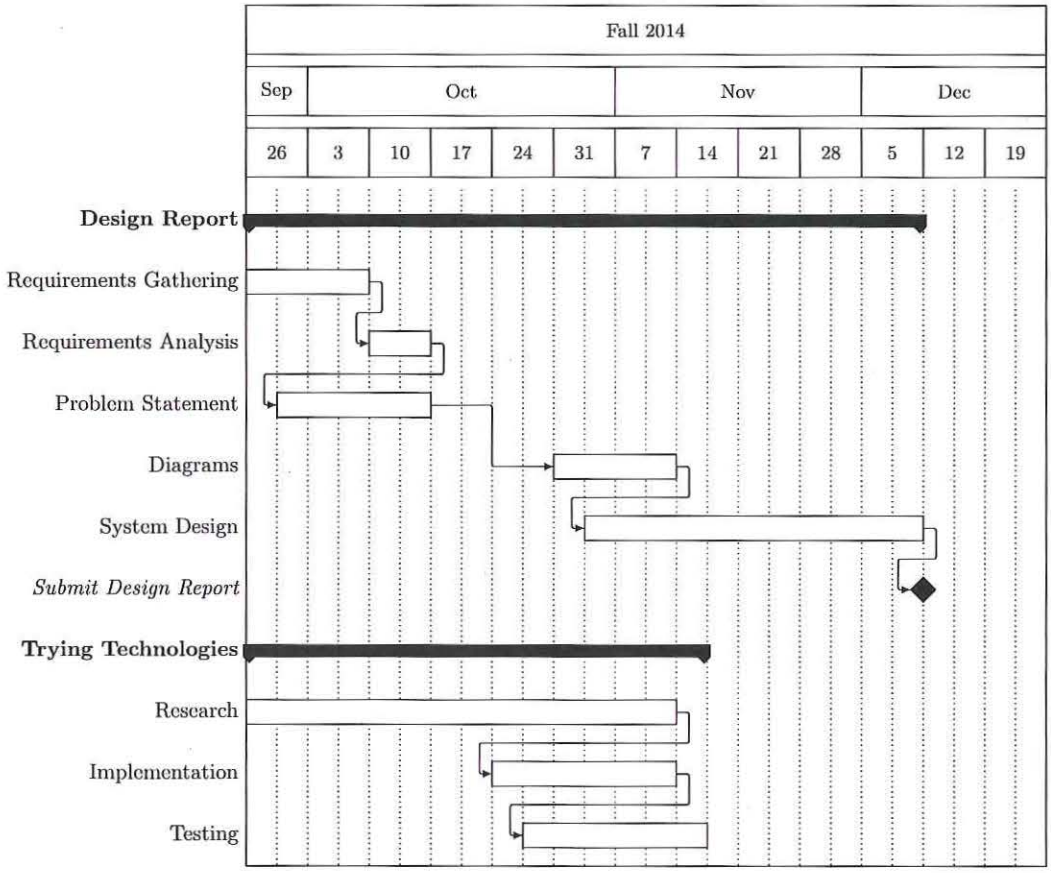

Figure 10.1: Fall quarter gantt chart

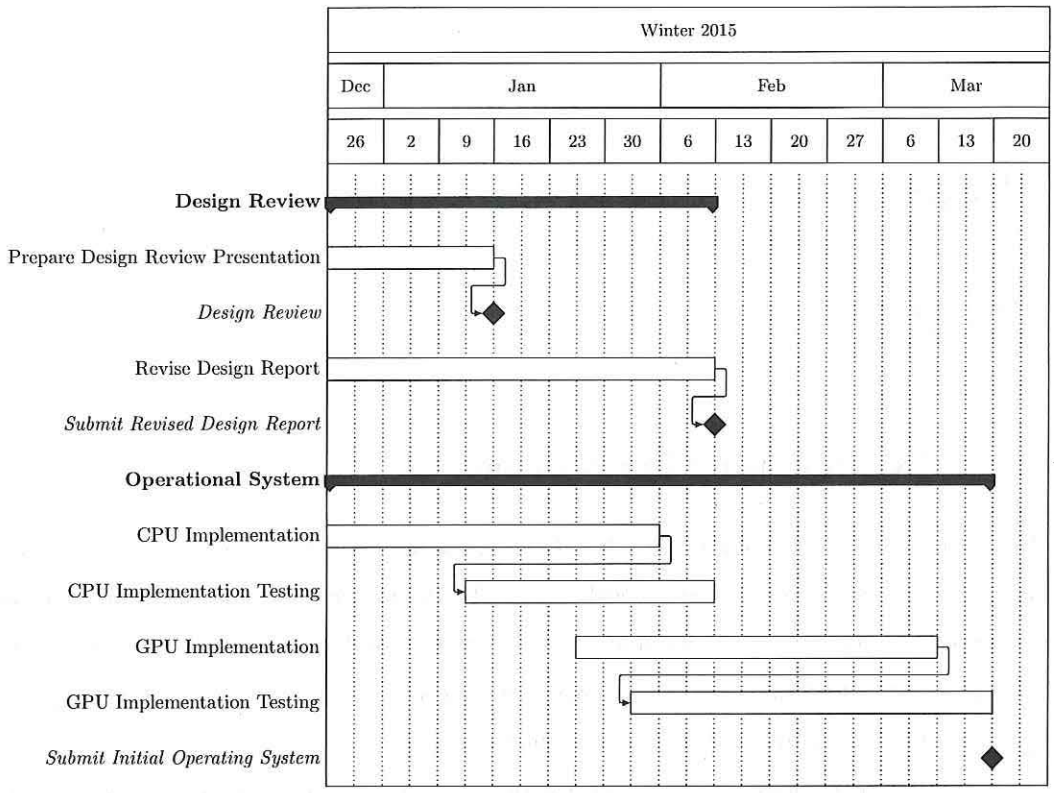

Figure 10.2: Winter quarter gantt chart

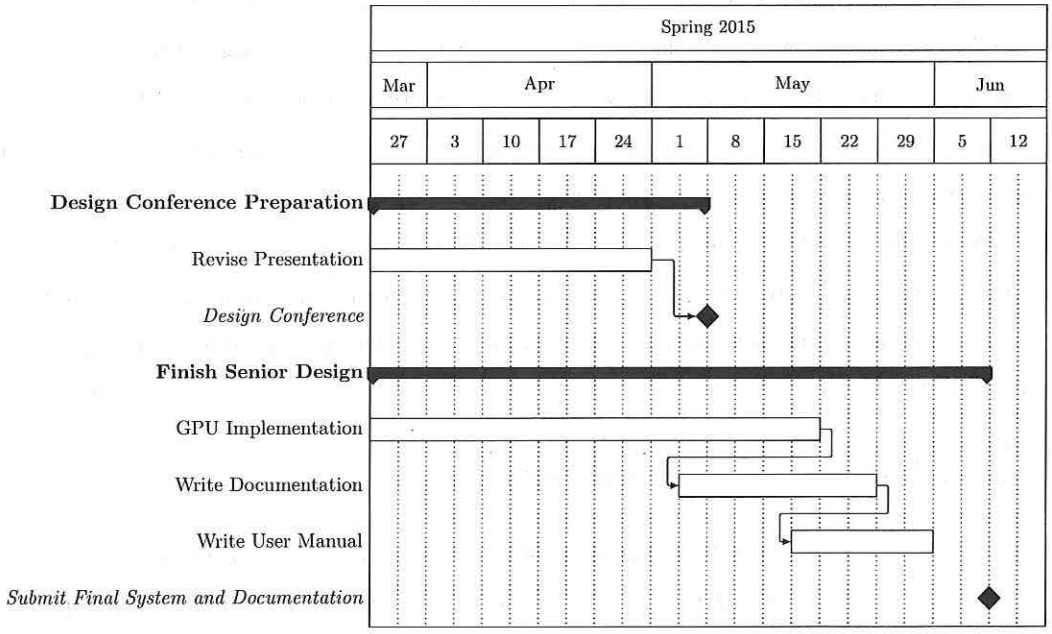

Figure 10.3: Spring quarter gantt chart

Chapter 11

Results

11.1 Pre-processing Input Images

Our initial plan was to use a geometric based and appearance based approach to pre process the input images before doing the shape modeling step. The geometric based approach involves doing edge detection on the image, which can highlight the contours of the lips better for easier detection. The appearance based approach involves doing color segmentation on the image which can separate the lips from the rest of the face due to their different color.

After trying these approaches, we found that they are not viable methods of enhancement. The edge detection alone did not slow down the program noticeably (it only added 10 milliseconds of processing time per frame) but it made detection much worse. The shape modeling algorithm could not find landmarks on the input image after an edge detection filter was applied to it. The color segmentation filter did not worsen the shape modeling algorithm like edge detection did, but it also did not noticeably improve shape modeling either. Also, as a downside, doing color segmentation alone takes a noticeable amount of time (40 milliseconds per frame), so it slowed down the program considerably. And finally, combining the edge detection with color segmentation simply resulted in having both of their negative effects. As a result, we have decided that have no pre-processing step is the best choice for this application. The current architectural diagram for the system can be seen in Figure 7.2.

11.2 Mouth vs Face Detection

Initially, we tried to detect and track only the mouth in the video feed. To do this, we used a statistical model that only had mouth points and a Haar Cascade design to detect a mouth. This process sometimes worked, but it ended up being too inaccurate since the Haar Cascade often narrowed the image down to the wrong area during the initial detection attempt. The initial detection is a crucial

part of this tracking method, so it was important for us to find a way to make sure it succeeded as often a possible.

Given the inefficiency of focusing on only the mouth, we then tried detecting the entire face with a full face model and a face Haar Cascade. This proved to be far more accurate than our attempts to detect the mouth by itself. This is because a face has more distinct landmarks which allow for more accurate detection. Once the detection finishes, we just throw away the points that are not relevant to the lips.

11.3 Face Detection Issues

Since we are detecting an entire face, instead of just the lips in the video frames, some problems can occur depending on the user. For example, users with glasses can slightly skew the detection, since the model cannot find the eyes. However, this rarely causes problems with detecting the lips in the frames. One issue that does causes problems with lip detection is facial hair. If the user has a mustache or a beard, it is very likely that the lip contours will be wrong. The detector will think the top of the mustache is the upper lip.

11.4 GPU-Complications

We encountered a number of complications when attempting to convert our CPU code into a GPU implementation. The first difficulty is that transferring image data between the CPU and GPU is a fairly expensive procedure. When we first got the code running on the GPU it was running at less than one frame per second because it took so long to transfer images. We were able to move some more of the code onto the GPU so that there would be fewer image transfers over all, which brought the speed back up to a measurable rate of around 6 frames per second on the Tegra K1, but this still is not anywhere near being an improvement on the CPU speeds.

We found that the reason the GPU implementation is slower than the CPU implementation is due to the matchTemplate function. We use this function for each patch to fit the model to the target image. Usually, this function would be faster on the GPU since it tries to match the patch in every location on the target image simultaneously. However, our CPU implementation is highly optimized and for each 11x11 pixel patch, matchTemplate only searches a 16x16 pixel area. Thus, it is much faster to run matchTemplate on the CPU since it is only searching a few locations.

Lastly, the version of OpenCV with which we were working also did not support certain functions that we needed for and ideal implementation and we were forced to write our own. One example of

this is the gemm (general matrix multiplication) function used for performing matrix multiplication on the GPU.

11.5 Conclusion

Despite the issues we have on the GPU, we have a CPU implementation that runs at an average of 12 frames per second on the Tegra K1. We believe that this speed will prove to be sufficient for the language learning software for which we have been working on this lip-tracking library.

Chapter 12

Audience Analysis

Our audience consisted of three groups. The first was technical and very knowledgeable of our research area. The second were those who were there to judge us on our writing and oral presentations. The final group was comprised of those with little to no technical knowledge, including the people who might use the final application that our library runs in as well as our family and friends that attended to see us present.

12.1 Judges

The judges of our presentation were the most important members of our audience, as they were, after all, the ones who inevitably decide how well we do. They probably had a basic understanding of general engineering concepts, but most likely not any knowledge specific to our project's field. As a result, we had to explain the concepts to them in a way that they could understand. A combination of concise text and graphics was useful in this case.

12.2 Technical

The main member of this group was our project advisor. She had the strongest technical understanding of the material that we presented. Others with her experience and understanding of the subject matter were in the audience as well, such as our advisor's colleagues or other professors. Members of this audience group likely understood the subject matter of our presentation as well as we do if not better.

12.3 Users and Family/Friends

The people using the language learning application that our library powers want it to always work reliably. If it doesn't they will be upset and complain. As for family and friends, they wanted to

support us and try to learn about what we made. Both parties had very little to no knowledge of how our system works, so explanations to them needed to be very simple and straightforward, otherwise we risked boring or confusing them. Pictures, rather than text were incredibly beneficial here.

12.4 Conclusion

Much of our presentation was be targeted toward the judges, since they have the most impact on our success. We started out general by giving a simple, interface and functionality-based overview of the whole system. This allowed the people who didn't want a lot of complicated information to at least understand what our project is about. Once the general information was covered, we went into more information for the judges and other, more technical, audience members. Given that the judges were the primary focus of our presentation, we did not spend too much time on the most heavily technical sections so that we did not lose their attention in extensive computation and field specific jargon.

Chapter 13

Ethical Analysis

13.1 Ethical Justification for the Product

Every day, advances in technology are increasing the connectivity between people and cultures around the world. This technology provides a means by which anyone around the world could conceivably connect with anyone else. One of the inherent barriers to this process, however, is the fact that not all people speak the same language. In fact, there are thousands of different languages spoken around the world.

The main ethical justification for our product is based on its use with the language learning software for which we are creating it. Ideally, the way our product is used with the language learning application will help people to learn to improve their ability to speak additional languages. This increase in linguistic ability is important because being able to speak multiple languages increases a person's capacity for connecting with others in the world. More people becoming connected in this way fosters a greater sense of global community, which in term can lead to greater global moral awareness. Our hope is that the software we are making can help to bring humankind one step closer to a more socially connected world.

Its use in this language learning software is by no means the only application for our lip-tracking software. There are many other ways that lip-tracking software that is both accurate and fast could be useful. For example, one might use it to design lip reading software that tries to interpret what someone is saying just from seeing his or her mouth move. With such software it would be easier to do automated captioning of video so that blind people might be able to understand what is going on. With translation software this could be expanded to work for people who do not speak the language of the video as well.

13.2 Team and Organizational Ethics

First and foremost, we placed a high priority on being ethical to each other by each performing a fair share of the work. It is unfair for one person to have to do a majority of the project by himself. We clearly planned out which of us will take the lead on which portions of the project so that this will not happen. If one of us become ill or otherwise indisposed, the other would step in to pick up the slack, but the sick member would do his best to make up for the lost time once he felt better.

In addition to our ethical obligations to each other, the two of us also had an ethical obligation to our advisor and the rest of the team working on the full product. We made sure to keep our advisor well-informed of our progress throughout the year. This way she could better aid us if ran into trouble along the way. We also did not lie about our progress if we fell behind, as this would only hurt us, as well as everyone else involved in the project.

Another ethical issue is plagiarism. It is easy to copy code off the internet and pass it off as one's own without citing the author. We did not do this as it is not only ethically wrong but incredibly easy to avoid. We simply cited the author of any useful library or snippet of code that we found and utilized for our lip-tracking code library.

We also did not waste Santa Clara University's resources. To do this, we minimized the requirements of our project to the point where we did not need any funding at all.

Lastly, all of our actions complied with the IEEE/ACM code of ethics [3]. This means that we did work in the best interests of our advisor, as as well as in the public's best interest. Our code was written in the highest quality possible and was not copied unjustly from others. And finally, by making our code open-source, we are allowing others to use and learn from our work for their own projects.

13.3 Product and Society/Politics

Overall, our project should prove helpful to society by serving to increase people's linguistic diversity. We can, unfortunately, see a nefarious use for our lip-tracking library, which is its use in surveillance applications. It would technically be possible for our project to be applied in lip-reading software that could be used on people through CCTV cameras and figure out what they are saying. The Criminal Law Handbook states that we should not expect privacy in a public place [2]. Most people, however, expect that their conversations will be private when others are not around them. Despite this, we do not believe that the ethical fault lies within our product, but rather in the way that people might choose to use it. After all, GPU-accelerated lip-tracking algorithms don't spy on

people; people spy on people.

Since our product handles images of people's faces, a concern some people may have is that it might collect and send the photos to a malicious third-party. To combat this, our project's code will also be completely free and open-source. This means that people using our library can feel safe that our code won't do anything undesirable. For example, our code will not collect images or other data of the user and it will not send any data to a remote server. Additionally, open-sourcing the code allows others to benefit from our code by learning from it and using it for their own projects.

During beta-testing of our project, we may require having some people test out the software to see how it performs for various faces. We will only store photos of users if they explicitly give us permission to do so and if the photos will help us improve our algorithm. These photos will also not be given out online as a part of the open-source code. We will provide tools that can train the lip-tracker from a collection of photos that the developer will already have procured, but we will not provide any photos ourselves since this may infringe on the privacy of those in the photos.

13.4 Conclusion

We believe that our project adheres to all ethical guidelines by which we, as engineers and human beings, base our decisions. The sources of these guidelines include the IEEE/ACM code of ethics, the Markkula Center's Framework for Thinking Ethically, and Santa Clara's Engineering Handbook. Our project has the potential to benefit people in many different ways, the amount of which is only limited by human kind's technological ingenuity.

Chapter 14

Societal Issues

14.1 Economic

This lip-tracking library consists of CUDA, OpenCV, and C++ code. Each of these technologies is completely free to use. Additionally, we are making our code free and open-source, so anyone can use it. This puts no economic burden on the creation or use of our lip-tracking library.

14.2 Health and Safety

Our lip-tracking library does not pose any health and safety risks. It is designed to be used while stationary and speaking words in a regular manner. This, by itself, should not cause any injury to the user.

14.3 Manufacturability and Usability

We have designed our system to be very easily modified and rebuilt. We have automated CMake scripts that do the entire build process for the developer using our library, regardless of what operating system they are using. Additionally, our API is simple and straightforward to use, making our library accessible to even novice programmers.

14.4 Sustainability

We based our code on CUDA and OpenCV. These libraries not only continue to be improved, but they both are always backwards-compatible. This means that our library will always work in the future with the latest versions of CUDA and OpenCV. We also used C++ as our main programming language, which will continue to be supported on all the operating systems we are targeting. As a result, our library can either be left alone from now on or continue to be improved over time and it will always be compatible with its applications.

14.5 Environmental Impact

Our lip-tracking library has no perceivable environmental impact. It is designed to be used on the student's mobile device, so no new materials should be used or manufactured to use our system.

14.6 Lifelong Learning

During the creation of this lip-tracking library, we have learned a great deal. We learned the intricacies of CUDA and OpenCV and how to better debug code as well as work on code with improvement in mind rather than development. These lessons will certainly help us once we are working in the industry. Additionally, with the knowledge of these new technologies, we can make better decisions on how to approach solving problems in the future and opened a new programming paradigm to us for future study.

14.7 Compassion

The main goal of this project is to help students learn new languages. There are, however, many more potential uses for this lip-tracking library. For example, it can be used to help the deaf with lip reading. This system definitely has potential to open up many possibilities for reducing the suffering in the world.

Chapter 15

Aesthetics Analysis

15.1 Audience and Interface

Our portion of the project does not have a standard windowed user interface as it is simply a library of code to be used in other applications, such as the language-learning mobile application. For this reason, our aesthetics analysis will focus on how our code is written and presented in documentation. Further, the only probable audience that needs to be considered as far as the aesthetics is concerned is that group of people that would be looking at the code. For this reason, the aesthetics can be targeted entirely toward those with enough technical knowledge to at least partially understand programming and software.

15.2 Documentation

The first is elements of our documentation. It must be easy to read and understand by a variety of audiences. We also created several diagrams within the documentation. These diagrams are aesthetically pleasing to view as well as easy to understand. For example, lines within flow diagrams do not cross unnecessarily and objects are arranged in a logical fashion.

Commenting is also a major concern whenever one writes any amount of code. Code becomes exponentially more difficult to understand the more that is written. Comments serve as documentation within the code itself. Proper commenting can help to reduce the amount of time the reader must spend trying to figure out what parts of a body of code do. We have aimed to write descriptive comments, specifying what various portions of our code that might be difficult to ascertain normally do. We did not, however, comment on every little part of our code. Excessive commenting can often lead to just as much confusion as no commenting at all.

15.3 Code Simplicity and Clarity

The second place where aesthetics come into play in our project is in our code. When writing code there are several aesthetic properties to keep in mind. The code should not be overly complicated, which means that it should be as simple and straightforward as possible while still being correct and accomplishing the desired goal. Code should also be split up into useful modules. If code is all written in one place rather than being divided up, it creates monolithic blocks of code that are close to impossible to understand. These considerations lead to elegant and understandable code. Additionally, each module should have logically related functions, providing a high level of cohesion in the overall system. Further, unrelated sections of the code should not reference each other, as it makes it harder for a user to understand what is going on by unnecessarily increasing the coupling of the system.

15.4 Code Syntax

Our code needed to be written in a consistent and visually appealing way. There are many considerations to make when writing code so that it is visually appealing and easy to read. We used consistent and proper tabbing. Tabbing assists the reader in following which code falls into what subsection and how it relates to other lines of code. Another consideration is the spacing between variables, operators, and other portions of the code. For example, x = x + 5 is more visually appealing and easier to read than x=x+5. Lastly, adding lines of white space between sections of the code helps the reader to identify where one section of the code stops and another begins.

15.5 System Interface

As our final consideration, we made our library's interface simple and easy to use. The interface of a library of code are those functions which someone intending to use the library would call in order to utilize our code. The goal of our code is to take an image of a face and output a set of points outlining the person's lips. Rather than make the person using our library have to learn how our code works to use it, we have a function with an input of an image and an output of a list of points. If they want to delve deeper and be able to change some of our settings, we provide access to those as well, but the core functionality is be very easy to use.

Bibliography

[1] Daniel Baggio. *Mastering OpenCV.* Packt Pub., 2012.

[2] P. Bergman and S.J. Berman-Barrett. *The Criminal Law Handbook: Know Your Rights, Survive the System.* Criminal Law Handbook. Nolo, 2008.

[3] Don Gotterbarn, Keith Miller, and Simon Rogerson. Software engineering code of ethics. *Commun. ACM*, 40(11):110–118, November 1997.

[4] Ahmad B. A. Hassanat. Visual speech recognition. *CoRR*, abs/1409.1411, 2014.

[5] Jian Li, Yuqiang Lu, Bo Pu, Yongming Xie, Jing Qin, Wai-Man Pang, and Pheng-Ann Heng. Accelerating active shape model using gpu for facial extraction in video. In *Intelligent Computing and Intelligent Systems, 2009. ICIS 2009. IEEE International Conference on*, volume 4, pages 522–526, Nov 2009.

[6] Sbastien Stillittano, Vincent Girondel, and Alice Caplier. Lip contour segmentation and tracking compliant with lip-reading application constraints. *Machine Vision and Applications*, 24(1):1–18, 2013.

[7] Jinwei Wang, Xirong Ma, Yuanping Zhu, and Jizhou Sun. Efficient parallel implementation of active appearance model fitting algorithm on gpu. *The Scientific World Journal*, page 13, 2014.

[8] Mau-Tsuen Yang, Zhen-Wei You, and Ya-Chun Shih. Lip contour extraction for language learning in vec3d. *Mach. Vision Appl.*, 21(1):33–41, October 2009.

[9] Ling Zhang, Ming Zhang, and Heng-Da Cheng. Color image segmentation based on neutrosophy. *Optical Engineering*, 51(3):037009–1–037009–11, 2012.